Mentor® Embedded Linux® System Builder User and Reference Manual

For the COMe P2020

Software Version 3.0

January 2011

Contractor/manufacturer is:
Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/
Send Feedback on Documentation: supportnet.mentor.com/user/feedback_form.cfm

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction to System Builder

The following key sections are covered in this chapter:

System Builder Overview

System Builder is a powerful command-line tool that you can use to create Linux distributions (primarily for embedded devices) in an automated and reproducible environment. System Builder leverages the OpenEmbedded project to provide the user with a polished build environment to cross-compile and package software for a target device.

The OpenEmbedded project is essentially a repository of metadata used to develop a collection of "recipes" used by the BitBake utility (an open source build tool, http://docs.openembedded.org/bitbake/html/). These recipes consist of the source URL of the package, dependencies, and compile or install options.

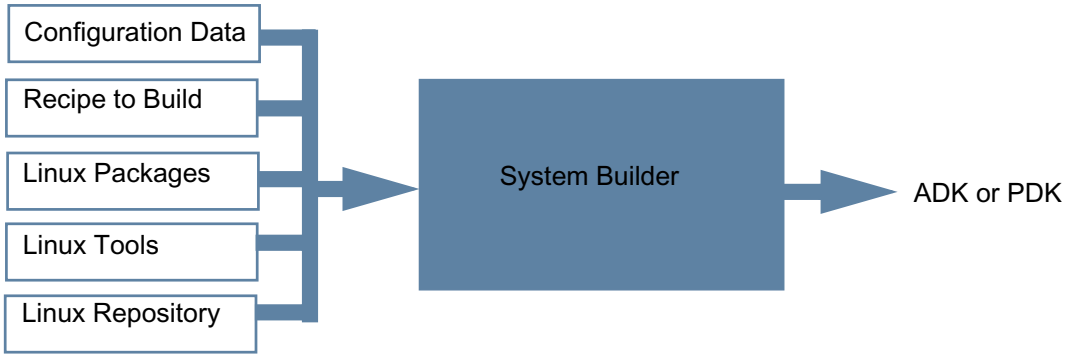
The System Builder utility contains a set of metadata used to cross-compile and package software packages into either one of the following objects (see Figure 1-1).

- Platform Development Kit (PDK): The PDK contains all that you need to create an entire system or platform, allowing you to rebuild the supported environment using caches, as well as being able to rebuild everything from source:
  - System Builder
  - Recipes
  - Sources for kernel, toolchain, and user space applications and libraries
- Application Development Kit (ADK): The ADK provides all software needed to develop applications to be used in the platform, and is also intended for quick evaluation of hardware. The ADK typically contains the following:
  - Linux kernel image
  - File system images
  - Host-runnable command line toolchain for one processor architecture

- Support libraries
- Firmware (U-Boot, and so on)

The ADK is a stand-alone environment that can be given to developers that do not need the entire System Builder environment. This allows for end-users to be able to quickly start evaluation of the hardware and allows for application developers to be able to work in a way that is more familiar to them and without adding new complexity to their workflow.

Figure 1-1. System Builder Diagram

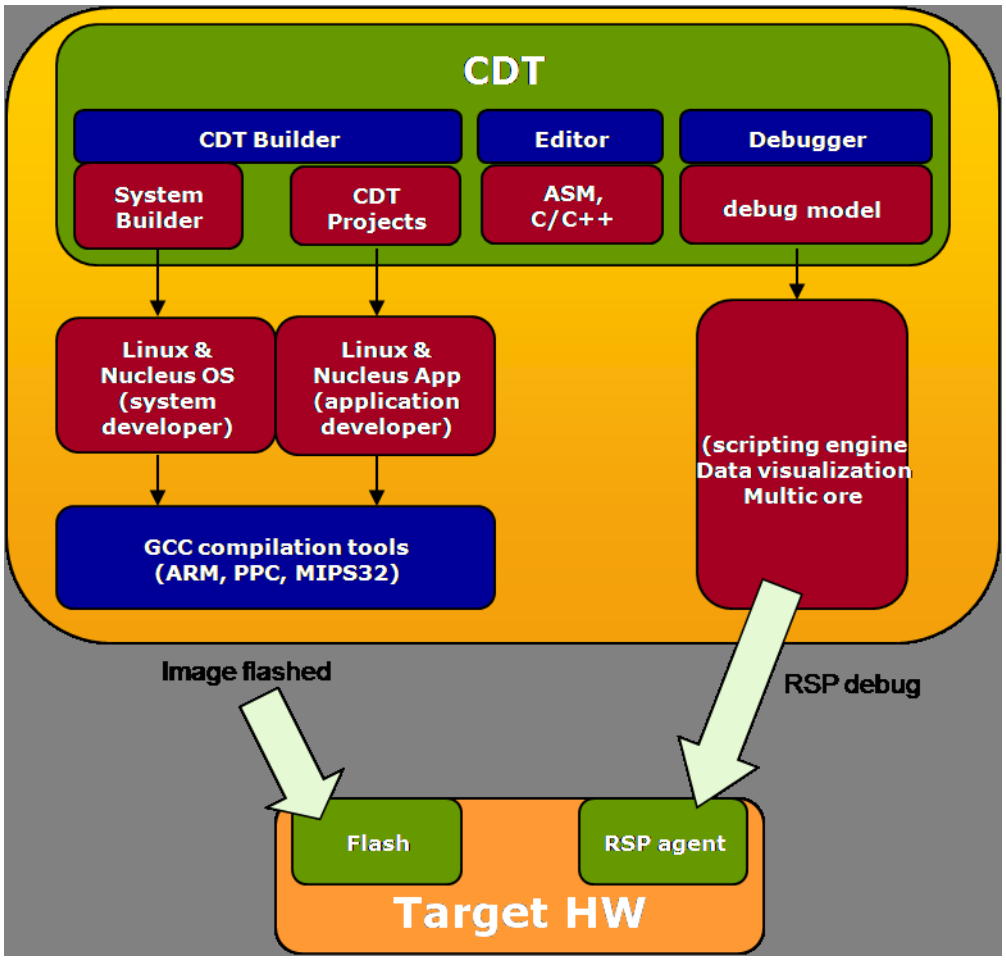


System Builder is comprised of the following:

- Metadata
- Sources or patches associated with the metadata
- Binary cache to shorten build times
- System builder tools

Aside from providing the open-source OpenEmbedded and BitBake utilities, System Builder is integrated into a wider Mentor Graphics toolchain (cross-compiler, editor, and debugger) to help you develop any application for a target hardware (see Figure 1-2).

Figure 1-2. System Builder in the Embedded Linux Toolchain



While System Builder leverages the OpenEmbedded project, it significantly enhances the ease of use and furthers reproducibility of the outputs, as well as providing several advantages and value-added features not available with open-source, including:

- **Easy Configuration Management**: System Builder provides a configuration tool that will, based on the parameters passed to it, create a build configuration file as well as environmental setup files. This greatly reduces the initial learning curve for any new user getting started with a build, as well as making it easier to create reproducible builds.
- **Configuration Knobs**: System Builder uses a feature known as “knobs” to determine the contents of our file system image and Application Development Kit (ADK). Each hardware target contains a list of supported features (or knobs). This is used by the configuration tool to validate a requested knob or to determine what knobs to prompt for. Based on these knobs, System Builder utilizes certain tasks (a kind of recipe) that decide what packages end up in the resulting output (file system image or ADK). This makes it easier to make sure that when the user starts up Linux on a target, there is enough software available to evaluate all of the capabilities of the hardware.

- **Caching of Binaries**: Building systems with OpenEmbedded typically require that all binaries are rebuilt from source on every run. System Builder enhances OpenEmbedded by allowing you to cache and reuse binaries from a previous build, thus reducing the time required to generate objects. SB ships with pre-built binary caches for the hosts that are supported.

System Builder Key Concepts

- **Collection**: A collection is a directory that contains an additional directory structure that matches what BitBake expects when it searches for metadata. As a best practice, when you add proprietary software to an ADK or PDK, you should create your own collection instead of mixing the binaries together with other applications in existing directories.
- **Metadata**: Metadata describes your configuration, descriptive information about upstream software, and instructions about how to build them.
- **Platform**: This term describes all software required for one customer project. For example, if a customer is building a router, then all software needed for the router product (including Mentor Graphics, customer, or third-party-supplied applications) is in the platform. This includes tools, sources and utilities. A platform contains objects such as:
  - User applications (libraries, applications, rootfs layout)
  - Kernel (source, binaries, and so on)
  - Firmware
- **Project**: This is the basic structure in which System Builder creates a particular output, either an ADK (for hardware evaluations or applications development) or PDK (creating the full platform). See "Project Scenarios" on page 26 for details on different types of System Builder projects.
- **Recipe**: A BitBake recipe is the metadata describing what tasks need to be performed to build an output object, including retrieving application source code, applying patches, providing additional files, compilation, installation, and generating the final binary package. The output is a binary package that is installable on the target hardware (plus intermediate files, such as libraries and headers).

System Builder Prerequisites

The following are required to run System Builder:

- **Host OS support** — A Linux host (real or virtual machine) running any of the following:
  - CentOS 5

  - RedHat Enterprise Linux 5
  - Ubuntu 8.04
  - Ubuntu 9.10
- **sudo access** — All Linux hosts require that the user to have “sudo” access. You must have access to the package manager through sudo.
- **Hardware requirements** — 30 GB of local disk space (approximate).
  - Network file systems such as CIFS and NFS are not supported due to locking issues.
  - The file system used by VMware for sharing folders works, but is not recommended due to it operating between 2 and 3 times as slow as storage local to the virtual machine.
  - 4 GB system memory.
- **Licensing** — Refer to the *Mentor Embedded Linux Product Installation Instructions* for complete details.
- **Required files** — Mentor Graphics provides all the setup scripts that need to be run in each installation. For Ubuntu hosts this is located in **scripts/ubuntu-oe.sh**, and for RHEL/CentOS, they are in **scripts/rhel5-oe.sh** (the **scripts** directory is in the PDK installation for all hosts).

Syntax Conventions

Table 1-1 identifies the syntax conventions used in this manual.

Table 1-1. Syntax Conventions

Convention	Description
Bold	A bold font indicates a required item.
Italic	An italic font indicates a user-supplied argument.
`Monospace`	A monospaced font indicates a shell command, line of code, or URL. A bolded monospaced font identifies text to be entered by the user.
Underline	An underlined item indicates either the default argument or the default value of an argument.
UPPercase	Uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword.
[ ]	Square brackets enclose optional arguments. Do not include the brackets when entering the command.

Table 1-1. Syntax Conventions (cont.)

Convention	Description
{ }	Braces enclose arguments to show grouping. Do not include the braces when entering the command.
‘ ’	Single quotes enclose metacharacters that are intended to be entered literally.
|	The vertical bar indicates an either/or choice between items. Do not include the bar when entering the command.
…	An ellipsis follows an argument or group of arguments that may appear more than once. Do not include the ellipsis when entering the command.

Chapter 2
Using System Builder

The following key sections are covered in this chapter:

The System Builder Workflow

The Mentor Embedded Linux System Builder builds upon and substantially simplifies the standard workflow for OpenEmbedded to create your embedded system.

An OpenEmbedded project typically requires that you manually create a configuration file using a text editor, then pass it to be built by the BitBake utility.

System Builder provides a configuration tool that, based on the parameters passed to it, create a local.conf file as well as environmental setup files. In addition, the configuration tool can be used to set up certain variables that require and the Mentor Graphics configuration “knobs” that are enabled. This greatly reduces the initial learning curve for any new user getting started with a build, as well as making it easier to create reproducible builds.

Prerequisites

- Installed the Mentor Embedded Linux software (refer to the *Mentor Embedded Linux Product Installation Instructions* for details) for the Platform Development Kit. This should include copies of OpenEmbedded and BitBake.

  For more information on OpenEmbedded and BitBake, refer to http://docs.openembedded.org/bitbake/html/.

Procedure

1. In the directory where System Builder has been installed (specifically in the /tools/pdk/<*machine*>/scripts directory), run the preparation scripts provided with the Mentor Graphics software. For Ubuntu hosts, they are located in **scripts/ubuntu-oe.sh**, and for RHEL/Centos, they are in **scripts/rhel5-oe.sh**.

```
./scripts/rhel5-oe.sh
```

2. Create directory structures called "collections" to store any proprietary applications (for example, *install-directory*/tools/{adk or pdk}/<*machine*> such as "*install-directory*/tools/pdk/p2020" and *install-directory*/tools/adk/p2020). Refer to "Using Collections" on page 19 for information.

```
mkdir -p <collection_name>/recipes/<application_name>
mkdir -p <collection_name>/recipes/<application_name>/files
```

   For example, to create a collection directory named *mycoll*:

```
mkdir -p mycoll/recipes/myapplication
mkdir -p mycoll/recipes/myapplication/files
```

3. Use the **./scripts/create-config.py** program to create your build directory. This program allows you to make selections for the build and create your configuration files. See also "create-config.py" on page 36.

```
./scripts/create-config.py -m <machine> -c <machine_collection_dir> -c
<other_collection_dir> -r -s <version>
```

   For example, to create a build directory from the PDK directory ./pdk/p2020, that contains a user-specified collection, *mycoll*:

```
  ./scripts/create-config.py -m p2020 -c fsl-come -c mycoll -r -s test -j2
-t2
```

   In this example, the required -m option specifies the hardware (or machine). The -c option references a "collection" directory. You must always specify the "machine" collection directory for hardware support, followed by any other -c collections you wish to add. The -j and -t options are additional parameters that can decrease run time as they specify parallel processing (-j determines number of jobs for make program to spawn, -t determines threads). Conservatively, the amounts specified can equal the number of processors available (for example, if you have a dual processor core, you could specify -j2 and -t2).

   Execution of this command line creates a build directory with environment variable files. For example:

```
build_p2020_release_test/bitbake.rc
build_p2020_release_test/bitbake.csh
```

   The -r option specifies that this is a "release" build (you can alternatively specify -d for "debug), and the -s option specifies the version.

You can also create abuild directory using the following nomenclature:

```
build_machine_build_version
```

The build indicates the build type (-r specifies “release”, -d specifies “debug”). Inside this directory are environment variable files (**bitbake.rc** and **bitbake.csh**) and the generated local.conf file, the build configuration file used by BitBake:

```
build_machine_release_version/local.conf
```

4. Setup your shell environment using **bitbake.rc** or **bitbake.csh**, depending on your command shell. The environment file (bitbake.rc or bitbake.csh) must to be executed prior to building a project. If you change projects, the bitbake file must be loaded into the environment. The “source” command is a good way to set up the environment variables. For example:

   ```
   source build_p2020_release_test/bitbake.rc
   ```

5. Execute BitBake to build the target(s). Depending on what output you would like to produce:

   - To build an ADK (minimum development kit to pass to other developers):

     ```
     bitbake standalone-environment-linux
     ```

   - To build a PDK (full software platform):

     ```
     bitbake systembuilder-release
     ```

   Using BitBake is further detailed in “Building A Project” on page 23.

Results

You create an ADK or PDK that an be passed to other developers or potentially to the hardware target.

Related Topics

Building A Project

create-config.py

Using Collections

Creating a PDK Project

The create-config.py script allows you to generate the configuration file to automatically manage the large number of settings and environment variables utilized for the build. The script simplifies this process by:

- Setting the build from the command line: The create-config script has a number of short-cut command line options that will help you create the appropriate build quickly.
- Using “knobs” to enable/disable common target features: The create-config script will automatically ask if you want to enable or disable a list common features found on the target board such as USB, hard disk, MTD, and so on.
- Caching and reusing the binary outputs of a build on a different machine of the same host OS and CPU class.

Note

Because large portions of physical memory are typically shared among multiple applications, the standard measure of memory usage known as resident set size (RSS) will significantly overestimate memory usage in the PDK. For more information, see http://www.selenic.com/smem/ .

Generating the Configuration File

You can invoke the create-config script from the command line with a wide variety of different options available. This section describes the basic use model for the create-config script.

For example, a user wants to evaluate a target board, “p2020”. The user needs to create a configuration file for the BitBake build that can collect the minimum firmware and applications needed to test the target. In this case, the create-config.py script is invoked with the following options:

```
./scripts/create-config.py -m p2020 -c fsl-come -r -s test
```

where:

- The -m option identifies the machine or target hardware (the p2020 board in this example).
- The -c option specifies a collection for the p2020 target board that Mentor Graphics provided with the System Builder installation. If you add your own collection, you would need to specify -c again. For instance:

```
./scripts/create-config.py -m p2020 -c fsl-come -c mycollection \
   -r -s test
```

- The -r option specifies that the distribution type is Release.
- The -s option specifies the version. In this case, the version is “test.”

See create-config.py for a full description of the command line syntax.

Once invoked, the script begins processing, and asks the following hardware-based questions:

```
Enable 'host-diskdrive' feature? [Y/n]
```

```
Enable 'host-usb' feature [Y/n]
Enable 'mtd' feature ? [Y/n]
Enable 'pci' feature ? [Y/n]
```

These are the "knobs" that you can select to enable or disable on the target. This list is dependent on the hardware. For more information on "knobs", refer to "Using Knobs" on page 20. The local.conf output of the create-config script (assuming the user selected "yes" for all the features) is:

```
MACHINE = "p2020"
DEVEL_FEATURES = "host-diskdrive host-usb mtd pci"
DISTRO = "systembuilder"
DISTRO_TYPE = "release"

PARALLEL MAKE = "-j 1"
BB_NUMBER_THREADS = "2"

HOSTVENDOR = "ubuntu_9.10_i686"
COLLECTIONS = "${OEDIR}/fsl-come ${OEDIR}/sb-core
      ${OEDIR}/openembedded"
TMPDIR = "{TOPDIR}"
DL_DIR = "{OEDIR}/sources"
DEPLOY_DIR_PSTAGE "${OEDIR}/cached-builds
build p2020 release test/conf/local.conf (END)
```

For this configuration file:

- MACHINE was set by the -m option.
- DEVEL_FEATURES was set by the "knobs" asked at the end of the configuration script run.
- DISTRO_TYPE was set by the -r option.
- OEDIR is the location of your installation (OpenEmbedded).
- "COLLECTIONS = ${OEDIR}/fsl-p2020" was set by the -c option. The "sb-core" collection is a default used by System Builder.
- PARALLEL_MAKE specifies how many make jobs can be run in parallel ("make" is a build tool used to compile software, so PARALLEL_MAKE affects the compile speed of an individual recipe). BB_NUMBER_THREADS specifies the maximum number of BitBake threads that can be launched. These are both set to minimum defaults (essentially a single processor core that can handle 1 make job and 2 threads). Further information on setting parallel Bitbake threads and make jobs can be found in the create-config.py reference page in Chapter 3, "System Builder Reference."
- TMPDIR specifies a temporary build directory (the default is the top level of your current working directory).
- DL_DIR specifies where source files that are downloaded or checked out from an SCM are to be placed (the default is "${OEDIR}/sources"). A different download directory can be specified using the --sources-download-directory option.

- DEPLOY_DIR_PSTAGE specifies where to find and to place all cached binary files that are used (and created). The default location is "${OEDIR}/cached-binaries." You can specify a different location by using the --cached-binary-directory option.

Full descriptions of the command line syntax and configuration file variable settings can be found in Chapter 3, "System Builder Reference."

By default, when building directly OpenEmbedded, everything generated resides in a subdirectory called "tmp." However, when using create-config instead, the directory is uniquely named, using the following syntax:

```
build_machine_build_version
```

where machine is the *name* of the target board, *build* is the build type (debug or release) and *version* is set by the -s option. The local.conf file generated is located in a "conf" subdirectory. For example:

```
./scripts/create-config.py -m p2020 -c fsl-come -r -s test
```

This command generates a "build_p2020_release_test" directory. The local.conf file is located in "build_p2020_release_test/conf". There are also environment variable files (**bitbake.csh** and **bitbake.rsh**) that you can source to set up the BitBake run.

The generated local.conf file is then used by BitBake to build the image to place on the target hardware.

Using Pre-Built Caches

Mentor Graphics provides the ability to cache and reuse the binary outputs of a build on a different machine of the same host OS and CPU class (for example, all 32-bit x86 Ubuntu 8.04 hosts). The caches, located in the "cached-builds" directory, are the output from builds that allow you to short-cut the lengthy process of setting up variables and dependencies for environment or target hardware. These caches are relocatable, which means that while performing a build, the cache can be placed anywhere you require, rather than an absolute location.

Use the --cached-binary-directory option for the create-config.py script if you decide to place cached builds in areas other than the default location. This sets the DEPLOY_DIR_PSTAGING variable to where the script will find and to place all cached binary files that are used (and created).

More information on managing cached binaries can be found in "Cached Binaries Management" on page 43.

Using Collections

A collection is a unique directory structure that BitBake uses to build an application. There are several default collections already present in an ADK or PDK.

It is recommended that if you wish to add a new application to an ADK or PDK, you should create a new collection for it rather than appending an existing directory, particularly if you are distributing binaries of proprietary software rather than open sources. You don't need to create a new collection for **each** new application you create; simply create them to have a collection of your own for your applications, rather than placing it within the collections provided.

For example, we want to add a “hellodemo” application to an ADK for PowerPC target running Linux. The source code is as follows:

```
#include <stdio.h>
int main(int argc, char **argv) {
   printf(“Hello World\n”);
   return 0;
}
```

The makefile is:

```
%.c%.o: %.c
   $(CC) $(CFLAGS) -o %.o -c %.c

all: hellodemo

hellodemo: hellodemo.o
   $(CC) $(LDFLAGS) -o $@ $@.o

install: all
   install -d $(DESTDIR)/usr/bin
   install -m 0755 hellodemo $(DESTDIR)/usr/bin

clean:
   rm -f hellodemo *.o
```

The program was then built using make, then cross-compiled for Linux PowerPC so that it is compatible with the target board. The sources are packed into a compressed file, hellodemo-1.0.tar.gz, for delivery. When the application is ready for deployment:

1. Create a directory structure in the PDK project location compatible with BitBake. For instance:

   ```
   mkdir -p ourcollection/recipes/hellodemo
   mkdir -p ourcollection/recipes/hellodemo/files
   ```

   In this example, “ourcollection” is the collection directory for the hellodemo application.

2. Copy your sources and recipe into the directories.

   ```
   cp hellodemo-1.0.tar.gz ourcollection/recipes/hellodemo/files
   ```

3. Create a BitBake recipe file for the application (the metadata that specifies that this is the software known to Bitbake). For example, the following is the recipe file, ourcollection/recipes/hellodemo/hellodemo_1.0.bb.

```
DESCRIPTION = “Hello world”
PR = “r0”
SRC_URI = “file://${PN}-${PV}.tar.gz”

do_compile() {
   oe_runmake
}
do_install() {
   oe_runmake DESTDIR=${D} install
{
```

   In this BitBake recipe, the ${PN} variable is the package name, ${PV} is the package version, and ${D} is the image or destination directory. Refer to the *BitBake User's Manual* for a complete description of recipe file syntax.

4. Refresh the configuration file by using create-config.py.

```
./scripts/create-config.py --config-file=build_.../.create-
config.ini -c hotfix
```

   You will be prompted that the build directory already exists.

```
Directory build_... already exists! Overwrite files and continue
anyway?
[y/n]
```

   Enter Y to continue. This re-creates the local.conf file to know about the hotfix collection and otherwise reuse all arguments given to create-config.py previously.

5. Build the image using BitBake.

Using Knobs

Knobs (sb-knobs.bbclass) are essentially toggles for high-level target board features. This feature relies on a mapping between a knob and a task (task-sb-alsa.bb) to determine what packages need to be installed (and in turn, what recipes make these packages). For a given machine, the list of all supported knobs is placed into the DEVEL_FEATURES_MAX environment variable of its *machine*.conf file (where *machine* is the name of the target), typically located in the “./*machine*/conf/machine” collection directory (for example, ./fsl-come/conf/machine). Refer to the section “Machine Configuration File” on page 44 for further information.

The list of all valid knobs is:

```
audio
bluetooth
host-diskdrive
```

```
host-usb
framebuffer
mtd
nas
native-development
pci
print-server
router
touchscreen
wifi
lttng
```

The actual supported list depends on your target board. By default, the create-config script automatically asks if you wish to enable each of the features supported, as in the example:

```
Enable 'host-diskdrive' feature? [Y/n]
Enable 'host-usb' feature [Y/n]
Enable 'mtd' feature ? [Y/n]
Enable 'pci' feature ? [Y/n]
```

However, you can also specify which features you want to enable from the command line invocation, using the following arguments:

- --enable-all-features: Enables all supported features for the target.
- --features = "*features*": Enables the feature(s) listed between the quotes. For example:

  ```
  ./scripts/create-config.py -m tgt12345 -c tgt12345 -r \
     -s test --features "mtd" --features "pci"
  ```

  This enables both the MTD and PCI features for the target board. The configuration script will not ask again at the end of the run. If you want to add features not listed, you can also use the -features command-line argument to enable it. For example:

  ```
  ./scripts/create-config.py -m tgt12345 -c tgt12345 -r \
      -s test --features "mtd" --features "pci" --features "router"
  ```

  This enables a "router" feature that is not on the "knobs" list. If this features is not supported for this machine, it will issue a warning:

  ```
  WARNING: The router feature is not officially supported
  ```

- --disable-all-features: Disables all supported features for the target.

Adding Your Own Recipes

The recipes that are used by the knobs determine the set of supported recipes (or packages of recipes) for System Builder. However, you can also add your own packages or additional open source packages by adding the following variables in the *machine*.conf file:

- PROJECT_EXTRA_PKGS: Add the contents of this variable into the devel-image file system.

- PROJECT_EXTRA_HOST_TOOLS: Add the contents of this variable into the ADK as additional host-side tools.
- PROJECT_EXTRA_DEVELOPMENT_PKGS: Add the contents of this variable into the ADK as additional target-side information (typically headers or libraries, but any recipe that builds for the target is valid).
- PROJECT_EXTRA_TOOLCHAIN_TARGET_EXCLUDE: Remove the contents of this variable from the ADK.

  For example, if you have myapp as a recipe which contains both "/bin/myapp" and "/usr/include/myapp/interface.h," adding myapp-dev to PROJECT_EXTRA_DEVELOPMENT_PKGS will pull myapp (containing /bin/myapp) into the ADK. If you put "myapp" into this variable, the ADK will contain (assuming PROJECT_EXTRA_PKGS contains myapp) a file system image with /bin/myapp and an ADK that contains ".../usr/include/myapp/interface.h" but not ".../bin/myapp."

When adding applications to an ADK or PDK, there are two basic use cases to keep in mind:

- The first case is adding an existing program that has corresponding metadata already provided (typically in the openembedded directory) but that is not part of the development image already. This is handled by using the PROJECT_EXTRA_PKGS variable.

  Where this variable is set depends on the use itself. For example, if you simply want to just try out a feature to see if it suits your needs, placing it into configuration file and then doing bitbake devel-image would be right.

  For example, the "u-boot" utility is placed into PROJECT_EXTRA_PKGS in the *machine*.conf file for all of the supported boards.
- The second example, which builds upon the first one, is that there is no corresponding metadata, which means that it must be created. There are two basic flows:
  - You can have existing programs that simply expect the user to invoke "make" and "make install" with a correct set of additional arguments as needed (for example, the hellodemo program shown in "Using Collections" on page 19 shows a basic example).
  - You can have an existing program that makes use of the GNU autotools suite. In this case, the recipe will have, rather than writing out do_compile and do_install functions, just "inherit autotools" which brings in the autotools.bbclass file defining how to configure, build, and install the program.

Building A Project

To build your system, you will use the BitBake tool to parse metadata and execute the commands described within that metadata. BitBake uses a configuration file (local.conf) as its instructions on how to build for your target.

The basic syntax for BitBake is:

```
bitbake target
```

where *target* is any recipe file that exists and is in the collections that have been passed to create-config.py for use. Each collection page lists the common high level targets that are valid. More specifically, ADKs and PDKs are built based on the standalone-environment-linux.bb and systembuilder-release.bb files, respectively:

- To build an ADK (minimum development kit to pass to other developers):

  ```
  bitbake standalone-environment-linux
  ```

- To build a PDK (full software platform):

  ```
  bitbake systembuilder-release
  ```

When processing a recipe, Bitbake will not repeat steps that have completed successfully before. If you wish to have a rerun a step, you may either use:

- Use -c clean *target* to remove the entire *target* work area. Note that this is not recursive and does not clean items that *target* depends on.
- Use -f -c *stage* to force that stage to be re-run.

To see a dependency graph for *target* rather than building it, pass the -g flag.

For complete information on BitBake, refer to http://docs.openembedded.org/bitbake/html/.

The following is an example session:

```
$ ./scripts/ubuntu-oe.sh
Now we're going to install all the other development packages needed to
build OE, please wait
[sudo] password for user:
Reading package lists... Done
Building dependency tree
...
Finally, we are going to install bitbake.
running build
running build_py
creating build
creating build/lib.linux-i686-2.6
creating build/lib.linux-i686-2.6/bb
copying lib/bb/COW.py -> build/lib.linux-i686-2.6/bb
...
```

```
Done. You're ready to go with OE build now
$ ./scripts/create-config.py -m p2020 -c fsl-p2020 -r -t6 -j8 \
   --enable-all-features --use-local-mirror-first --local-mirror \
   http://mirror.vendor-internal.com/sources
Finished.
$ . ./build_p2020_release/bitbake.rc
$ cd build_p2020_release
$ bitbake standalone-environment-linux
...
NOTE: Tasks Summary: Attempted 2539 tasks of which 1765 didn't need to be
rerun and 0 failed.
```

Amending a Project

So you have a completed build, but you want to change something in it, perhaps a code correction or add an additional application. To add changes to an existing project, you create an amend.inc file. This is an include file that contains instructions for any updates or patches for your collection.

This utilizes the amend.bbclass functionality found in OpenEmbedded. When create-config.py is run, it will parse the amend.inc file found in a given subdirectory. For example, with openembedded/recipes/*vendorapp*, any recipe found in there will also read sb-core/recipes/*vendorapp*/amend.inc. Because of this, amend.inc should be written in such a way as to be specific as possible about when to apply the changes found in there. The amend.inc file exists once per recipe.

For example, let's say that an application called "hellodemo" (previously defined in "Using Collections" on page 19) needs an update. We want to change the string for "Hello World" to "Hello Demo." The new code would be:

```
#include <stdio.h>
int main(int argc, char **argv) {
printf ("Hello Demo")
}
```

Let's assume we have an initial collection containing the recipe for the initial version of hellodemo:

```
ourcollection/recipes/hellodemo/hellodemo_1.0.bb
```

A patch file, hellodemo-1.0.patch, is then created for distribution. To add the patch:

1. Create a new directory (for this example, we'll call it "hotfix") with a recipes/hellodemo subdirectory.

   ```
   mkdir hotfix/recipes/hellodemo
   ```

   In this case, "hotfix" isn't just a directory, it's a collection that you'll need to add to your build (you would need to re-run create-config.py with "-c hotfix" before running BitBake again).

2. Copy the patch over to the new location.

```
cp hellodemo-1.0.patch hotfix/recipes/hellodemo
```

3. Create the hotfix/recipes/hellodemo/amend.inc file.

```
PR.= “.1”
# The .1 indicates that we’ve made a single change

SRC_URI += “file://hellodemo-1.0.patch;patch=1”
# This instructs BitBake to add in the patch file
```

4. Run the create-config.py script.

```
./scripts/create-config.py --config-file=build_.../
      .create-config.ini -c hotfix
```

   You will be prompted that the directory already exists.

```
Directory build_... already exists! Overwrite files and continue
anyway?
[y/n]
```

   Enter “Y” to continue. This will re-create the local.conf file to know about the hotfix collection and otherwise reuse all arguments given to create-config.py previously.

5. Save the file and rebuild.

```
bitbake hellodemo
```

This is a fairly basic example of using amend.inc and relies on the fact that inside of ALL “recipe/hellodemo” directories there exists only one recipe, hellodemo.

However, a more complex example involves larger numbers of recipes. For example, the openembedded/recipes/python directory contains well over 100 different recipes (.bb files). There is also a single sb-core/recipes/python/amend.inc file. By default, it means that the contents of that amend.inc file will be applied to each of those recipes, which doesn’t make sense. The amend.inc can thus be written to selectively apply overrides. For example, let’s say that we only need to update recipes with “python-native” as part of its file name:

```
SRC_URI_append_pn-python-native =
"file://python-2.6-r73342.patch;patch=1;pnum=0"
PARALLEL_MAKE_pn-python-native = ""
PR_append_pn-python-native .= ".1"
```

While each of those recipes will contain, for example:

```
PR_append_pn-python-native .= ".1"
```

when the PR variable is evaluated, the previous code will only be used when “pn-python-native” is in the list of overrides that BitBake uses (refer to http://docs.openembedded.org/bitbake/html/ for a complete description of override) and pn-

python-native is only in the list when the recipe name is “python-native” (“pn-${*PN*}” is always in the list of variable overrides).

Project Scenarios

If you consider the typical workflows needed to create a complete platform for target hardware, the roles and objectives of each of the flows can be met by using System Builder to create specific “projects” to generate ADKs or PDKs. This section will provide detailed descriptions of how to create System Builder projects and illustrate usage examples for each scenario.

The Turbo Router Example

This section will illustrate the scenarios through the use of a fictional “Turbo Router” series of projects to create three routers implemented by an imaginary company called Beech Networks.

In this scenario, Beech Networks plans to share some libraries and applications but yet be unique enough to address different market segments. They will be adding their own intellectual property (IP) in various portions of each of their platforms.

Table 2-1. “Beech Networks” Turbo Routers

Router Name	Description	Family/Device
Low	Consumer electronics, home use functionality	ARM11 / MX27
Mid	SOHO, mid-level features	PowerPC e300 / MPC8372
High	Enterprise router, high-end, advanced features	PowerPC e500 / P4080

The development flow for this series of platforms involves several main paths (these flows are illustrated in Figure 2-1).

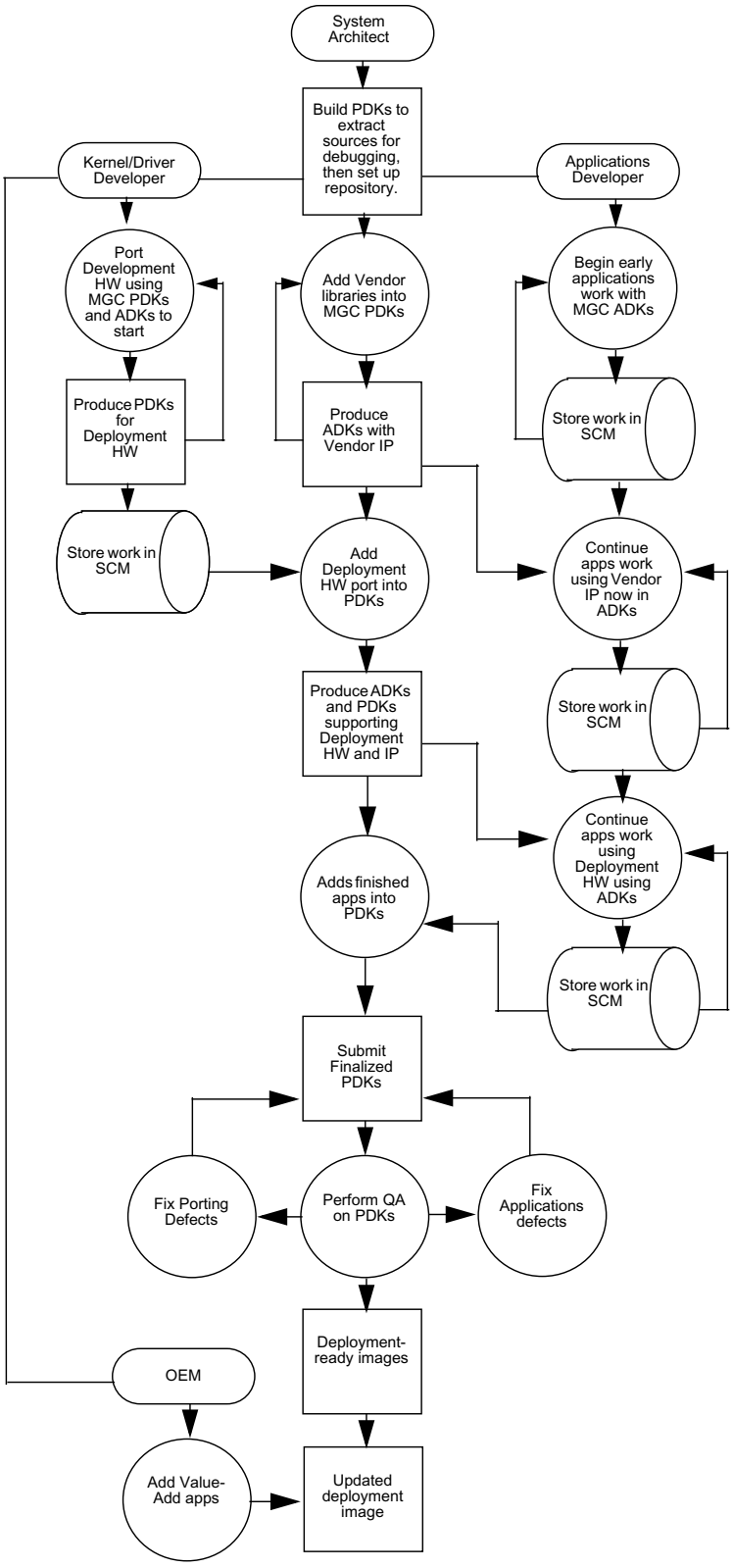
- **System Integrator/Architect**: A System Integrator/Architect and architect starts out by adding common libraries and environmental items to the PDKs and will be the main person creating and distributing internal ADKs and PDKs. The System Integrator’s path continues by integrating in the port to the deployment hardware and debugged applications. The System Integrator/Architect eventually makes the final deployment images.
- **Applications Developer**: An Applications Developer can start his work as soon as he has access to the ADKs. As the Applications Developer continues through the application development cycle, updated ADKs are passed from the System Integrator containing Beech Network-specific IP libraries and, eventually, kernels for the deployment hardware. In the end, the Applications Developer gives the final debugged applications to the system integrator for inclusion into the vendor deployment images.

- **Kernel and Driver Developer**: The K&D developer uses a PDK as a starting point to develop firmware and Linux ports to the Beech Networks deployment hardware. These drivers and firmware may also include some of Beech Network's IP. When finished, the K&D Developer gives the code to the System Integrator/Architect for inclusion into the vendor's deployment images.
- **OEM**: After the vendor product is deployed, an end-customer may wish to add additional applications to the final product. Another project can be generated to build the system with these additional applications.

Because System Builder and the Mentor Embedded Linux IDE have ties to software configuration management (SCM) tools, the easiest way to pass code would be through an SCM repository (though simple copy or linking of files can also be used).

To address having a series of three different router platforms, Beech Networks would have to address each of the routers independently as separate platforms. However, the System Integrator can use the SCM tools to manage common pieces without duplication of code.

Figure 2-1. System Builder Project Flows



As shown in Table 2-2, it is conceivable that the following projects would be created and used to produce a platform.

Table 2-2. Project Scenarios for Beech Networks

Role	Project	Description
System Integrator	Beech Networks IP	Add Beech Networks IP to Mentor PDKs/ADKs
	Beech Networks Deployment	Add Deployment HW port to PDKs/ADKs
	Beech Networks Final Image	Add final applications and create final deployment images.
Kernel and Driver Developer	Deployment HW Port	Port firmware, kernel and drivers for Deployment hardware
Applications Developer	Early Application Development	Develop applications using a Mentor ADK and reference hardware targets
	Application Development	Develop applications using a Beech Networks custom ADK with Beech Networks IP and ref hardware targets
	Deployment Application Development	Develop applications using Beech Networks custom ADK with Beech Networks IP and Deployment HW port using Deployment hardware
OEM	Deployed Image Value-Add	Add new non-Beech Network applications to released Beech Networks image.

System Integrator Flow

The project is described as having three router platforms, low, mid, and high. For the purposes of this example:

- The router platforms will be referred to in this example as tr-low, tr-mid, and tr-high for hardware platform names.
- In all cases (tr-low, tr-mid, and tr-high routers), the steps will differ only in which machine is referenced. This section will only describe the tr-mid platform.
- The System Integrator is using Ubuntu 9.10 as the Linux host.

Following installation, the steps that the system integrator will perform are as follows:

1. Create a directory that will become the collection for the router.

   ```
   mkdir beech-turborouter
   ```

2. Add the required configuration subdirectory for BitBake.

   ```
   mkdir -p beech-turborouter/conf/machines
   ```

3. Use a text editor to create a configuration file for the router.

   ```
   vi beech-turborouter/conf/machines/tr-mid.conf
   ```

   At this point, when creating each of the machine configuration files, the System Integrator will have set variables to:

   - Describe the platforms
   - Enable feature knobs
   - Use existing open source packages that already have recipes needed to complete the project

For example, given that the mid-level Turbo Router uses the PowerPC e300, the machine configuration file might look as follows:

```
#@TYPE: Machine
#@Name: Mid Turbo Router
#@DESCRIPTION: Machine configuration for the Mid Turbo Router

TARGET_ARCH = "powerpc"

#Change the appropriate lines to reflect the version of U-Boot and the
#U-Boot config target of your machine

KVERSION = "2.6.25"
KERNEL_IMAGETYPE = "uImage"
PREFERRED_PROVIDER_virtual/kernel = "linux"

PREFERRED_VERSION_u-boot = "1.3.3"
UBOOT_MACHINE = "tr-mid_config"
UBOOT_ENTRYPOINT = "0"
UBOOT_LOADADDRESS = "0"

MACHINE_FEATURES = "kernel26 usbhost pci ext2 uboot"
DEVEL_FEATURES_MAX = "host-diskdrive host-usb mtd pci router"
PROJECT_EXTRA_PKGS = "u-boot"

#tune for the e300c3 core
require conf/machine/include/tr-mid-ppce300c3.inc
```

The System Integrator can now create the ADK that the K & D Developer and Applications Developer will use to add Beech Networks Intellectual Property (IP) for the Turbo Router.

1. Run the create-config.py script.

```
./scripts/create-config.py -r -m tr-mid --enable-all-features \
   --jobs 3 --threads 2
```

2. Source the BitBake settings.

   ```
   source build_tr-mid_release/bitbake.rc
   ```

3. Run BitBake to build the platform.

   ```
   bitbake standalone-environment-linux
   ```

   This creates the following file:

   ```
   build_tr-mid_release/deploy/glibc/delivery/adk/\
      standalone-environment-Ubuntu_9.10_i686-ppce300c3-tr-\
      mid-linux.tar.bz2
   ```

   This compressed file contains an ADK and can now be sent to the K & D Developer and Applications Developer to begin their projects.

Applications Developer Flow

The Applications Developer, once given the initial ADK from the System Integrator, can start working on the Beech Networks applications and libraries that contain their IP. The basic steps are as follows:

1. Create a work area for the ADK.

   ```
   mkdir ~/work/tr-mid-adk-0001
   ```

2. Uncompress the ADK.

   ```
   tar -C ~/work/tr-mid-adk-0001 -xf \
      standalone-environment-Ubuntu_9.10_i686-ppce300c3-tr-\
      mid-linux.tar.bz2
   ```

3. Develop and debug the program using your favorite IDE.

4. Once the application is completed, the Applications Developer informs the System Integrator where the application is located to import for the next ADK. As shown in Figure 2-1, this is a fairly iterative process where applications can be updated as required during the life cycle of the process. Each major iteration has its own project as more pieces of the platform are added to the ADK (for instance, the hardware port).

   By the end of this cycle, the Applications Developer should have all recipe information for his project (all info necessary to checkout, build, dependencies, target file system locations, and so on) ready to be checked into SCM and have the platform rebuilt to incorporate the new applications.

Kernel and Driver Developer Flow

In this example, the K&D Developer is assumed to be more comfortable working from the command line.

Once the K&D Developer has been given an ADK by the System Integrator, work can begin on the Linux Kernel port for the router hardware platforms, which include writing the required drivers and other items which may or may not be Beech Networks IP.

The K&D Developer's steps are:

1. Copy the source from the PDK (the installed version used by the System Integrator/Architect) to a local directory. This allows the developer to use the PDK build tools.
2. Create a work area for the ADK. For example:

   ```
   mkdir ~/work/tr-mid-adk-0001
   ```

3. Uncompress the ADK. For example:

   ```
   tar -C ~/work/tr-mid-adk-0001 -xf \
      standalone-environment-Ubuntu_9.10_i686-ppce300c3-\
      tr-mid-linux.tar.bz2
   ```

4. Make sure that the PATH is updated for the new work area.

   ```
   export PATH=${PATH}:$HOME/work/tr-mid-adk-0001/bin
   ```

The K&D Developer then uses the installed ADK toolchain to develop code on their copy of the kernel tree source, using a preferred text editor to build the Linux Kernel from the command line. Refer to your kernel documentation for complete information on building kernels.

Once the code is working, the updated patches are submitted to the System Integrator/Architect for incorporation into the production build which can then yield a new PDK and ADK (which then can be distributed to other developers that require the kernel). In this example, the passing along of changes is done using an SCM.

Finalizing the Platform

Once the System Integrator has been given an updated Linux Kernel by the K&D Developer and the user applications and libraries from the Applications Developer, all of the updates must be integrated for System Builder to build a new ADK for final testing. To create this final, integrated ADK:

1. Edit to say what additional recipes need to be built as well as any changes to say what version of the kernel to use and how file system images are generated and so on …

   ```
   vi beech-turborouter/conf/machines/tr-mid.conf
   ```

2. Create a directory for recipes for the Beech Networks applications.

   ```
   mkdir -p beech-turborouter/recipes/bn-apps/
   ```

3. Create the recipe files for all the applications and libraries. Refer to http://docs.openembedded.org/bitbake/html/ for complete information on how to create recipes.

   ```
   vi beech-turborouter/recipes/bn-apps/firstlib_1.0.bb
   ```

   This should be done for all libraries and applications.

4. Create an amend.inc file to have the kernel be patched correctly as well as any further updates (refer to "Amending a Project" on page 24 for a description of the amend.inc file).

   ```
   vi beech-turborouter/recipes/linux/amend.inc
   ```

5. Build the project using BitBake.

   ```
   bitbake standalone-environment-linux
   ```

   This produces the following compressed file:

   ```
   mid_release/deploy/glibc/delivery/adk/standalone-environment\
      -Ubuntu_9.10_i686-ppce300c3-tr-mid-linux.tar.bz2
   ```

   This file is the final ADK that QA will perform testing on the kernel and file system images. After testing is complete, the platform is ready for release.

Adding to a Deployed Release

In addition, let's suppose that an OEM wishes to add content to a released ADK. In this case, the OEM would essentially follow the same process as the System Integrator (without the same handoffs to the other contributors), as well as having to make the ADK distributable.

Chapter 3
System Builder Reference

This chapter describes the full syntax of the System Builder utility, specifically the create-config.py command-line script. A quick reference page for the BitBake utility (fully documented in the *BitBake User's Manual*) is also included in this chapter.

create-config.py

This program allows you to make selections for the build and create your configuration files.

Usage

create-config {-m | --machine} ***machine_name***
[--help]
[{-d | --debug} | {-r | --release}]
[{-s | --version} *string*]
[{-t | --threads} *N*]
[{-j | --jobs} *N*]
[{-C | --collections-directory} *oe_dir*]
[{-c | --collections} *path*]
[--cached-binary-directory *pstaging_dir*]
[--sources-download-directory *dir*]
[--signature-policy {strict | loose]
[--list-available-features]
[--enable-all-features | --features "*string*" | --disable-all-features]
[{-l | --local-mirror} *url*]
[--use-local-mirror-first]
[{-p | --cached-binary-mirror} *url*]
[--config-file *filename*]

Arguments

- **{-m | --machine}** ***machine_name***

  A required argument that specifies the machine (hardware target). For example, if you have an sb-core/conf/machine/tgt12345.conf file, you would pass “-m tgt12345” as the argument. In the configuration file generated, this argument sets the MACHINE variable.

- --help

  Lists all supported command-line options.

- -d | --debug
  -r | --release

  Specifies the build type to be performed, either a debug or release. The default setting is -r. In the generated configuration file, these arguments set the DISTRO_TYPE variable.

- {-s | --version} *string*

  Specifies the version. This appends an identifier to the build directory so you can have multiple versions with the same machine and release/debug settings.

Parallel Processing

- {-t | --threads} *N*
  {-j | --jobs} *N*

  Sets the amount of parallel processing to be used. The -t or --threads option specifies the maximum number of BitBake tasks (or threads) that can be issued in parallel.

  The -j or --jobs option determines the number of jobs to have the make program itself spawn during the compilation stage.

  The process of building a specific target is typically broken down into a number of steps. When building, all of the dependent programs required as well as the substeps of each and their dependencies with each other are calculated, a task list is created, and then those tasks are executed.The -t and -j options allow you to have better control over parallel task and job processing when performing your BitBake builds (for instance, you can increase the limits if you are using a multi-core processor for faster builds).

  The default settings are -j 1 and -t 2. In the generated configuration file, these options set the PARALLEL_MAKE (for -j) and BB_NUMBER_THREADS (for -t) variables.

Collections and Cached Binaries

- {-C | --collections-directory} *oe_dir*

  -C or --collections-directory "path" to supply the directory which contains the collections. Relative paths supplied to -c are looked for here, as well as the default collections (sb-core, openembedded).

- {-c | --collections} *path*

  Lists additional collections to be used. This path may be absolute, or relative to the collections directory. By default, the sb-core and openembedded collections will be used. You can add more collections by adding further -c or --collections arguments (this is not a single-argument list with separators).

- --cached-binary-directory *pstaging_dir*

  Sets the DEPLOY_DIR_PSTAGING variable to where to find and to place all cached binary files that are used (and created).

Sources Download

- --sources-download-directory *dir*

  Sets the DL_DIR variable to where source files that are downloaded or checked out from an SCM are to be placed.

Signature Policy

- --signature-policy {loose | strict}

  Specifies the policy used for generation of a signature from the metadata. This signature is utilized in the cached binary file names. There are two available policies: loose and strict. When policy is set to “strict”, all variables that are not explicitly forbidden are included in

the signature. When policy is set to “loose”, only specified variables are captured in this signature. Using the strict policy is recommended, and is the default.

Feature-Based (Knobs)

System Builder uses a feature known as “knobs” to determine the contents of the file system image and Application Development Kit (ADK). A list of supported knobs for a given machine is placed into its DEVEL_FEATURES_MAX variable.

In turn, this is used by the configuration tool to validate a requested knob or to determine what knobs to prompt for. Based on these knobs, System Builder installs certain tasks (a kind of recipe) that decide what packages end up in the resulting output (file system image or ADK). This makes it easier to make sure that when the user starts up Linux on a target, there is enough software available to evaluate all of the capabilities of the hardware.

Note that the --enable-all-features and --features arguments are mutually exclusive.

- --list-available-features

  Prints the list of all supported features for the given machine and exits.

- --enable-all-features

  Sets the list of features to build to be the same as all supported features.

- --features "*string*"

  Specifies the features to enable. This argument sets the DEVEL_FEATURES variable in the configuration file. For any word in the *string* that is not in DEVEL_FEATURES_MAX, a warning message is printed. This option appends to a list and can be specified multiple times for multiple features.

Note

The quotes (") are required only if there are spaces in the string and if it is required by your command shell.

- --disable-all-features

  Disables all features to avoid prompting during the script run.

Download Mirror Settings

- {-l | --local-mirror} *url*

  Specifies the URL of the mirror to download sources from.

- --use-local-mirror-first

  Sets the specified source mirror to be used as a first choice to download from rather than adding to the list of places to download from, should the file not be found in the initial search.

- {-p | --cached-binary-mirror} *url*

  Specifies the URL of the mirror to download cached binaries from.

Loading From A Configuration File

- --config-file *filename*

  Loads options from a user-specified configuration file. In addition to the configuration file supplied through --config-file, the create-config.py script also automatically loads ~/.create-config.ini, for site-wide configuration options.

Description

This program creates the “local.conf” file that OpenEmbedded needs to finish the configuration of an environment. In addition, the configuration tool can be used to set up certain variables that require and the Mentor Graphics configuration “knobs” that are enabled. This greatly reduces the initial learning curve for any new user getting started with a build, as well as making it easier to create reproducible builds.

In addition to the config file supplied using --config-file, create-config.py also automatically loads ~/.create-config.ini, for site-wide configuration options.

The syntax of the file is that of an “ini”: key=value pairs within a section. Note that multi-line values are supported in RFC-822 style (second and subsequent lines indented by spaces). In the script’s usage of this format, multi-line values are used to implement lists, and the only option that uses it is the --collections option.

Any option that can be set using the command line may be set using the configuration file. The keys in the file are the command line option, long or short, without the leading dashes. For example:

```
[General Options]
debug = True
use-local-mirror-first = True
local-mirror = http://mirror.vendor-internal.com/sources/
cached-binary-directory = ${OEDIR}/prebuilt
sources-download-directory = ${OEDIR}/downloads
jobs = 8
threads = 6
[Target Feature Support options]
enable-all-features = True
```

Examples

```
create-config.py -r -m tgt12345 -c tgt12345 -s add_router -t6 -t8 \
   --features "mtd"

create-config.py -m p12345s -c fsl-p12345s -r -t6 -j8 \
   --enable-all-features --use-local-mirror-first --local-mirror \
   http://ea.vendorsource.com/sources/
```

Related Topics

Creating a PDK Project

bitbake

Executes the specified task for a set of BitBake files. The BBFILES variable must be defined, which is a space-separated list of files to be executed. Default BBFILES are the .bb files in the current directory.

Complete information on BitBake is in the *BitBake User's Manual*, distributed with this software package.

Usage

bitbake
[--version]
[-h | --help]
[-b *BUILDFILE* | --buildfile=*BUILDFILE*]
[-k | --continue]
[-f | --force]
[-i | --interactive]
[-c *CMD* |--cmd=*CMD*]
[-r *FILE*, --read=*FILE*]
[-v | --verbose]
[-D | --debug]
[-n | --dry-run]
[-s | --show-versions]
[-e | --environment]
[-I *IGNORED_DOT_DEPS* | --ignore-deps=*IGNORED_DOT_DEPS*]
[*package* ...]

Arguments

- --version

  Displays program version number and exit.

- -h | --help

  Displays list of supported arguments and exit.

- -b *BUILDFILE* | --buildfile=*BUILDFILE*

  Executes the task against this .bb file, rather than a package from BBFILES. Using -b to build a recipe short-circuits the dependency resolution process, so if the dependencies haven't already been built previously, it will fail.

- -k | --continue

  Continues the run as much as possible after an error. While the target that failed, and those that depend on it cannot be remade, the dependencies of these targets can be processed all the same.

- -f | --force

  Forces run of specified task, regardless of stamp status.

- -i | --interactive

  Enables interactive mode (BitBake shell).

- -c *CMD* |--cmd=*CMD*

  Specifies task to execute. Below is a partial list of tasks (*CMD*) you can execute:

  - clean — cleans the package. Does not touch the deploy directory
  - fetch — gets the package source from the source tree
  - patch — patches the source with the patches provided in the package
  - configure — configures the package
  - compile — compiles the package
  - build — builds the package
  - install — installs the package
  - package — packages the package
  - menuconfig <*kernel*> — runs make menuconfig on the kernel, this opens the console configuration tool for the kernel.
  - menuconfig busybox — runs make menuconfig on busybox, this opens the console configuration tool for the busybox.

- -r *FILE*, --read=*FILE*

  Reads the specified file before the bitbake.conf file.

- -v | --verbose

  Enables verbose output to the terminal.

- -D | --debug

  Increases the debug level. You can specify this more than once.

- -n | --dry-run

  Processes run without execution.

- -s | --show-versions

  Shows current and preferred versions of all packages.

- -e | --environment

  Shows the global or per-package environment.

- -g | --graphviz

  Displays dependency trees of the specified packages in the dot syntax

- -I *IGNORED_DOT_DEPS* | --ignore-deps=*IGNORED_DOT_DEPS*

  Stops processing at the given list of dependencies when generating dependency graphs.

- *package*

  Specifies the package name.

Examples

```
bitbake test
bitbake -c clean test /cleans the test package
```

Related Topics

Building A Project

Chapter 4
System Builder Best Practices

The following key sections are covered in this chapter:

This section describes how we recommend working with System Builder, given a particular set of circumstances. These topics assume a familiarity with how any of the relevant collections are implemented.

Cached Binaries Management

System Builder provides the ability to cache and reuse certain steps in the build process. An important part of this feature is making this cache be easily sharable between multiple users. This is implemented by allowing for a URI to be set as a location to fetch the cached file if it is not found in the local directory that BitBake expects it to be in. This URI is passed using the -p or --cached-binary-mirror options in the create-config.py utility.

There are several methods of managing cached binaries:

- HTTP / FTP: One simple way to manage the server is to store the files on an HTTP or FTP server and allow appropriate users to be able to add new files to the cache.
- Revision Control System: Another method is to store the cache inside of an RCS such as Subversion or git.

External Sources Management

Part of building System Builder involves using Open Source programs available over the internet. For any program that Mentor Graphics includes for a release, a copy of all of these sources are also included (located in the /sources directory). However, Mentor Graphics

provides the metadata for customers to build things that have not been prebuilt. Additionally, while Mentor may have chosen to use a specific version of a program at one point in time, in the future, the sources could move or disappear.

To work around this issue, System Builder (as part of OpenEmbedded) allows for mirror sites to be checked for source files. These mirror sites can either be checked first or last. Finally, these sites are only checked if a copy of the sources does not already exist on the host the build is being performed on.

Importing Linux BSPs

In some cases, support for a Linux Board Support Package (BSP) exists from a third party, in the form of a series of patches for the Linux kernel and the firmware (typically U-Boot). In OpenEmbedded terms, the term “machine” is used to describe a particular hardware platform (for example, the Freescale P2020DS). To allow for support using System Builder, the following are recommended actions:

- In some cases, there is a starting point found upstream (the openembedded directory) and in other cases, you would be starting from scratch. You should always check the upstream location first to see if there something to start with.
- As a general rule, when adding support for a machine, it should be placed into a collection of its own. This will allow for easy future maintenance, allows for support for merging upstream, and makes delivery of the support an opt-in rather than opt-out operation.

Machine Configuration File

The machine configuration file (*machine*.conf, where *machine* is the name of the target) must describe a number of objects such as the CPU architecture and other hardware specific variables. The file that is read here is set by the contents of the MACHINE variable. For many CPU architectures, there are include files that already exist and should be referenced that will setup various compiler flags. In addition, the following variables need to be set for a Linux kernel:

- PREFERRED_VERSION_virtual/kernel = "*VERSION*"
- KERNEL_IMAGETYPE = "*build_target_to_pass_to_make*"
- KERNEL_OUTPUT = "*location_of_resulting_kernel_image*"

Linux Kernel Support

Using the amend.inc feature, adding support for a Linux kernel can generally be done in a single file. This file must contain the following:

- SRC_URI_append_pn-linux =, " file://0001-this-first.patch;patch=1" if any patches are required for the Linux kernel version to be used.

In the "*collection*/recipes/linux" directory, a linux-VERSION directory (for example, linux-2.6.29) must be created and the kernel configuration file to be used should be copied there and named "defconfig." The amend.inc file always resides in a recipe directory and is used if amending to an existing in another collection recipe file (for example, *collection*/recipes/linux_2.6.29.bb). and not used if there is not a corresponding recipe (for example, openembedded/recipes/linux_2.6.40.bb doesn't exist, so it is usually better to create *collection*/recipes/linux/linux_2.6.40.bb rather than use amend.inc).

U-Boot Support

U-Boot is supported in a manner similar to Linux kernel support, using a single file through the "amend" functionality. This file must contain:

- SRC_URI_append_pn-u-boot =, " file://0001-this-first.patch;patch=1" if any patches are required for the Linux kernel version to be used.

In the "*collection*/recipes/u-boot directory", a u-boot-*VERSION* directory (for example, u-boot-2009.11rc1) must be created and inside of the machine.conf file. The following variables must be set:

- UBOOT_MACHINE must be set to the U-Boot config target name, if it is something other than ${MACHINE}_config.
- UBOOT_ENTRYPOINT and UBOOT_LOADADDRESS must be set if they need to be something other than 0x20008000.

SCM / RCS Integration

When applying software configuration management (SCM) or a revision control system (RCS), System Builder can be considered having two "sides":

- Metadata including recipes, classes and other non-binary data that is used to build things for a given target, on a given host. For metadata, for a multiuser or production environment, it is recommended that the metadata should all be placed into some form of revision control.
- External and internal sources used to make target software along with the binaries of these pieces of software. For management of internal sources, refer to "Cached Binaries Management" on page 43. For management of external sources, refer to "External Sources Management" on page 43.

Debugging System Builder Outputs

In certain cases, you may wish to debug output exactly as it is built by System Builder (BitBake). In this case, perform one of the following steps:

- Delete any previously existing files or changes by using the following command:

  ```
  bitbake -c clean recipe
  ```

- Compile the software.

  ```
  bitbake -c compile recipe
  ```

- Use your debugger to look in the *build_output_directory*/work/*build-architecture*/*recipe-version-pr*/*recipe-version*/ directory for all of the sources. Note that build-architecture may not always be what you expect as certain target packages are specific to the target hardware.

Example 4-1. Debugging System Builder outputs using a the *busybox* recipe and a P2020 machine

```
$ bitbake -c clean busybox
NOTE: Handling BitBake files: - (8053/8053) [100 %]
NOTE: Parsing finished. 7418 cached, 310 parsed, 325 skipped, 129 masked.
NOTE: Resolving any missing task queue dependencies
NOTE: Preparing runqueue
NOTE: Executing runqueue
NOTE: Running task 1 of 1 (ID: 0, .../tools/pdk/WHATEVER/sb-
core/recipes/busybox/busybox_1.13.2.bb, do_clean)
staging-busybox-p2020-linux-gnuspe - 1.13.2-r23.2.8IwWQLclz87r5jU_b1s3qA
Removing package staging-busybox-p2020-linux-gnuspe from root...
NOTE: Removing staging package ../cached-
builds/systembuilder/release/glibc/p2020-linux-gnuspe/staging-busybox-
p2020-linux-gnuspe_1.13.2-r23.2.pNYdNN4iSw89B2zRxKmGkg_i686-linux.ipk
NOTE: removing work/p2020-linux-gnuspe/busybox-1.13.2-r23.2
NOTE: removing stamps/p2020-linux-gnuspe/busybox-1.13.2-r23.2.*
NOTE: Tasks Summary: Attempted 1 tasks of which 0 didn't need to be rerun
and 0 failed.
$ bitbake -c compile busybox
NOTE: Handling BitBake files: - (8053/8053) [100 %]
NOTE: Parsing finished. 7418 cached, 310 parsed, 325 skipped, 129 masked.
NOTE: Resolving any missing task queue dependencies
NOTE: Preparing runqueue
NOTE: Executing runqueue
NOTE: Running task 276 of 285 (ID: 0, .../tools/pdk/WHATEVER/sb-
core/recipes/busybox/busybox_1.13.2.bb, do_setscene)
NOTE: Running task 279 of 285 (ID: 1, .../tools/pdk/WHATEVER/sb-
core/recipes/busybox/busybox_1.13.2.bb, do_fetch)
NOTE: Running task 280 of 285 (ID: 2, .../tools/pdk/WHATEVER/sb-
core/recipes/busybox/busybox_1.13.2.bb, do_unpack)
NOTE: Unpacking ../sources/busybox-1.13.2.tar.gz to work/p2020-linux-
gnuspe/busybox-1.13.2-r23.2/
...
```

```
NOTE: Running task 284 of 285 (ID: 5, .../tools/pdk/WHATEVER/sb-
core/recipes/busybox/busybox_1.13.2.bb, do_qa_configure)
NOTE: Running task 285 of 285 (ID: 7, .../tools/pdk/WHATEVER/sb-
core/recipes/busybox/busybox_1.13.2.bb, do_compile)
NOTE: Tasks Summary: Attempted 285 tasks of which 277 didn't need to be
rerun and 0 failed.
```

And in this case, the debugger checks build_p2020_release/work/p2020-linux-gnuspe/busybox-1.13.2-r23.2/busybox-1.13.2.

Example 4-2. Debugging System Builder outputs using a the *crashme* recipe

```
$ bitbake -c clean crashme
NOTE: Handling BitBake files: - (8053/8053) [100 %]
NOTE: Parsing finished. 7418 cached, 310 parsed, 325 skipped, 129 masked.
NOTE: Resolving any missing task queue dependencies
NOTE: Preparing runqueue
NOTE: Executing runqueue
NOTE: Running task 1 of 1 (ID: 0, /work/easi_oe/system-builder/release-
2010.1/sb-core/recipes/crashme/crashme_2.4.bb, do_clean)
staging-crashme-ppce500v2-linux-gnuspe - 2.4-r0.YD6RtnLWjt86HUxgZRwhgg
Removing package staging-crashme-ppce500v2-linux-gnuspe from root...
NOTE: Removing staging package ../cached-
builds/systembuilder/release/glibc/ppce500v2-linux-gnuspe/staging-
crashme-ppce500v2-linux-gnuspe_2.4-r0.mNMOVKsph1eLKIQNUROCGQ_i686-
linux.ipk
NOTE: removing /work/ppce500v2-linux-gnuspe/crashme-2.4-r0
NOTE: removing stamps/ppce500v2-linux-gnuspe/crashme-2.4-r0.*
NOTE: Tasks Summary: Attempted 1 tasks of which 0 didn't need to be rerun
and 0 failed.
$  bitbake -c compile crashme
NOTE: Handling BitBake files: - (8053/8053) [100 %]
NOTE: Parsing finished. 7418 cached, 310 parsed, 325 skipped, 129 masked.
NOTE: Resolving any missing task queue dependencies
NOTE: Preparing runqueue
NOTE: Executing runqueue
NOTE: Running task 276 of 285 (ID: 0, /work/easi_oe/system-
builder/release-2010.1/sb-core/recipes/crashme/crashme_2.4.bb,
do_setscene)
NOTE: Running task 279 of 285 (ID: 1, /work/easi_oe/system-
builder/release-2010.1/sb-core/recipes/crashme/crashme_2.4.bb, do_fetch)
NOTE: Running task 280 of 285 (ID: 2, /work/easi_oe/system-
builder/release-2010.1/sb-core/recipes/crashme/crashme_2.4.bb, do_unpack)
NOTE: Unpacking ../sb-core/recipes/crashme/files/crashme-2.4.tar.bz2 to
work/ppce500v2-linux-gnuspe/crashme-2.4-r0/
...
NOTE: Running task 284 of 285 (ID: 5, /work/easi_oe/system-
builder/release-2010.1/sb-core/recipes/crashme/crashme_2.4.bb,
do_qa_configure)
NOTE: Running task 285 of 285 (ID: 7, /work/easi_oe/system-
builder/release-2010.1/sb-core/recipes/crashme/crashme_2.4.bb,
do_compile)
NOTE: Tasks Summary: Attempted 285 tasks of which 277 didn't need to be
rerun and 0 failed.
```

In this example, the debugger looks in build_p2020_release/work/ppce500v2-linux-gnuspe/crashme-2.4-r0/crashme-2.4.

Rebuilding the Kernel Using the PDK

The Mentor Graphics PDK typically contains a prebuilt kernel image configured for an NFS-root development environment. It may become necessary to enable or disable additional kernel configuration options.

It is important to keep a clean copy of the kernel source tree. It is good practice to perform kernel builds from the your home directory by making a copy of the clean kernel source tree.

This section describes how to build the Linux kernel image using the PDK. Kernel developers will typically use the ADK and directly call make to build kernels, but this section explains how a System Integrator might rebuild the kernel using the PDK BitBake workflow.

Prerequisites

- Have configured (with create-config.py) the PDK for your build.

Procedure

1. Prepare the environment from the cache:

   ```
   bitbake virtual/kernel
   ```

2. Get a new, clean kernel tree. For example:

   ```
   bitbake -c configure virtual/kernel
   ```

3. Configure the kernel as follows:

   ```
   bitbake -c menuconfig virtual/kernel
   ```

   This spawns a new terminal window with the kernel configuration program open. If a new window cannot be spawned, edit build_.../conf/local.conf and add the following based on your environment (prior to issuing the bitbake command):

   For a non-X11 environment:

   ```
   TERMCMD = "${SCREEN_TERMCMD}"
   TERMCMDRUN = "${SCREEN_TERMCMDRUN}"
   ```

   Use **screen -r** to attach to the kernel configuration program from a different terminal.

   When using a GNOME-based X11 environment, you do not need to use the **screen -r** command. Instead, your terminal window appears automatically.

   The following commands can be used for the other environments:

   For a GNOME environment (default):

```
TERMCMD = "${GNOME_TERMCMD}"
TERMCMDRUN = "${GNOME_TERMCMDRUN}"
```

For a KDE environment:

```
TERMCMD = "${KONSOLE_TERMCMD}"
TERMCMDRUN = "${KONSOLE_TERMCMDRUN}"
```

For non-GNOME and non-KDE environments:

```
TERMCMD = "${XTERM_TERMCMD}"
TERMCMDRUN = "${XTERM_TERMCMDRUN}"
```

4. Build a new kernel, as in the following example:

```
bitbake -c compile virtual/kernel
bitbake virtual/kernel
```

Results

The Linux kernel image is found in build_.../deploy/glibc/images/. In this example, the image contains uImage-mpc8377e-rdb.bin and uImage-mpc8377e-rdb.dtb.

Using a Custom Kernel Configuration

It is relatively simple to build with a custom user-supplied kernel config using System Builder. The method outlined in this section will result in a non-volatile change to your kernel configuration. Simply editing defconfig or .config in your work directory will result in your kernel configuration being overwritten the next time something changes, and will cause the kernel to be rebuilt.

Prerequisites

- Have configured (with create-config.py) the PDK for your build.

Procedure

1. First, create your custom .config file. One possibility is to use devshell after you have finished a complete build.
2. After setting up your bitbake environment, execute:

```
bitbake virtual/kernel
bitbake -c configure virtual/kernel
bitbake -c devshell virtual/kernel
```

Or alternatively, you could issue:

```
bitbake virtual/kernel
bitbake -c configure virtual/kernel
bitbake -c menuconfig virtual/kernel
```

Both of the final commands (devshell and menuconfig) obey the rules set out previously with the various TERMCMD variables and their behavior (as defined in the previous section "Rebuilding the Kernel Using the PDK" on page 48).

3. From within the devshell terminal, type "make menuconfig". (Or, if you used -c menuconfig, you'll already be in the kernel configuration menu.)
4. Make your changes and save the *.config* to a different location, or it will be overwritten next time you build linux. The following example saves the *.config* file as *defconfig* in *${OEDIR}/my-collections*, as shown below:

```
mkdir pdk/WHATEVER/my-collection
cd pdk/WHATEVER/my-collection
~pdk/WHATEVER/my-collection$ tree
.
`-- recipes
    `-- linux
        `-- defconfig
```

5. Edit *local.conf* as follows:

```
COLLECTIONS = "${OEDIR}/my-collection ${OEDIR}/fsl-p1022ds
${OEDIR}/sb-core ${OEDIR}/openembedded"
```

6. Rebuild Linux using BitBake:

```
bitbake -c clean virtual/kernel
bitbake virtual/kernel
```

Results

These steps should result in your new kernel being built with your custom configuration.

Related Topics

Rebuilding the Kernel Using the PDK

bitbake

Using a Custom BusyBox Configuration

Building a Kernel Module Outside the Kernel Source Tree

The following is an example of how to build a kernel module (device driver) outside of the kernel source tree, often referred to as and "out-of-tree" driver or module. This example is within the context of using the System Builder PDK. You may perform this differently during active development/porting.

The following is a BitBake recipe, called lda-module.bb. This recipe includes a startup script, INITSCRIPT_* variables, and so on, so it isn't just a module, it has a user component (not all recipes will do so).

```
DESCRIPTION = "Linux Debugger Agent Kernel Module"
SECTION = "base"
PRIORITY = "optional"
LICENSE = "GPL"

INITSCRIPT_NAME = "lda"
INITSCRIPT_PARAMS = "defaults 21"

SRC_URI = "\
  file://Makefile \
  file://utils.c \
  file://utils.h \
  file://utils-x86.c \
  file://utils-ppc.c \
  file://utils-arm.c \
  file://syscalls.c \
  file://server.c \
  file://server.h \
  file://proto.c \
  file://proto.h \
  file://net.c \
  file://net.h \
  file://log.h \
  file://init.c \
  file://cmds.c \
  file://cmd-file.c \
  file://breakpoints.c \
  file://lda.init \
 "

S = "${WORKDIR}"

inherit module update-rc.d

PR = "r5"

do_install() {
   install -d ${D}${base_libdir}/modules/${KERNEL_VERSION}/extra/
   install -m 0644 ${WORKDIR}/lda.ko
${D}${base_libdir}/modules/${KERNEL_VERSION}/extra/

   install -d ${D}${sysconfdir}/init.d/
   install -m 0755 ${WORKDIR}/lda.init ${D}${sysconfdir}/init.d/lda
}

FILES_${PN} = "${base_libdir}/modules/"
FILES_${PN} += "${sysconfdir}/"
```

Using a Custom BusyBox Configuration

It is relatively simple to build with a custom user-supplied busybox configuration using System Builder. The method outlined in this section will result in a non-volatile change to your busybox configuration. Simply editing defconfig or .config in your work directory will result in your busybox configuration being overwritten the next time something changes, and will cause the busybox to be rebuilt.

Prerequisites

- Have configured (with create-config.py) the PDK for your build.

Procedure

1. First, create your custom .config file. One possibility is to use devshell after you have finished a complete build.
2. After setting up your bitbake environment, execute:

   ```
   bitbake busybox
   bitbake -c configure busybox
   bitbake -c menuconfig busybox
   ```

3. Make your changes and then save your .config to a different location, or it will be overwritten next time you build linux. The following example saves the .config file as “defconfig” in ${OEDIR}/my-collections, as shown below:

   ```
   mkdir pdk/WHATEVER/my-collection
   cd pdk/WHATEVER/my-collection
   ~pdk/WHATEVER/my-collection$ tree
   .
   `-- recipes
       `-- linux
           `-- defconfig
   ```

   Then edit your local.conf as follows:

   ```
   COLLECTIONS = "${OEDIR}/my-collection ${OEDIR}/fsl-p1022ds
   ${OEDIR}/sb-core ${OEDIR}/openembedded"
   ```

4. Finally, rebuild *busybox* using BitBake:

   ```
   bitbake -c clean busybox
   bitbake busybox
   ```

Related Topics

Using a Custom Kernel Configuration

bitbake

Index

Third-Party Information

- *Third-Party Software for Embedded Products*

Embedded Software and Hardware License Agreement

The latest version of the Embedded Software and Hardware License Agreement is available on-line at:
www.mentor.com/eshla

IMPORTANT INFORMATION

USE OF PRODUCTS IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF PRODUCTS INDICATES YOUR COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

EMBEDDED SOFTWARE AND HARDWARE LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Products (as defined in Section 1) between the company acquiring the license ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, if received electronically, certify destruction of Products and all accompanying items within five days after receipt of Products and receive a full refund of any license fee paid.

1. **DEFINITIONS.** These terms shall have the following meanings: "Customer's Product" means the end-user product identified in the applicable quotation which is developed solely by Customer; "Development Location" means the location where Software may be used as authorized in writing by Mentor Graphics; "Development Tools" means the software that may be used by Customer for building, editing, compiling, debugging or prototyping Customer's Product; "Embedded Software" means software that is embeddable; "End-User" means Customer's customer; "Executable Code" means a compiled program translated into a machine readable format that can be loaded into memory and run by a certain processor; "Hardware" means a physically tangible electro-mechanical system or sub-system and associated documentation; "Linkable Object Code" means linkable code resulting from the translation, processing, or compiling of Source Code by a computer into machine readable format; "Processor Unit" means the specific microprocessor to be used with Embedded Software and implemented in Customer's Product; "Products" means Software and/or Hardware; "Software" means software programs, Embedded Software and/or Development Tools, including any updates, modifications, revisions, copies, documentation and design data that are licensed under this Agreement; and "Source Code" means software in a form in which the program logic is readily understandable by a human being.

2. **ORDERS, FEES AND PAYMENT.**

   2.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement ("Order(s)"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.

   2.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will invoice separately. Unless provided with a certificate of exemption, Mentor Graphics will invoice Customer for all applicable taxes. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under such Orders in the event of default by the third party.

   2.3. All products are delivered FCA factory (Incoterms 2000), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

3. **GRANT OF LICENSE.**

3.1. All Products constitute or contain copyright, trade secret, proprietary and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. As set forth in the applicable quotation, Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software as set forth in the applicable subsection(s) in Section 3.2 below. The limited licenses granted under this Agreement shall continue until expiration or termination in accordance with Section 13 below, whichever occurs first. **Mentor Graphics does NOT grant Customer any right to (a) sublicense or (b) use Software beyond the scope of this Section without first signing a separate agreement with Mentor Graphics for such purpose.**

3.2. License Type. The license type shall be identified in the applicable quotation.

3.2.1. **Development License:** Customer may modify, compile, assemble and convert the applicable Embedded Software Source Code into Linkable Object Code and/or Executable Code form for the Processor Unit(s), Customer's Product(s) and at the Development Location(s) identified in the quotation, provided such Development Location(s) is located within the United States.

3.2.2. **End-User Product License:** In addition to the rights granted under Section 3.2.1, Customer may incorporate or embed an Executable Code version of the Embedded Software into unlimited copies of Customer's Product(s), using the Processor Unit(s), and at the Development Location(s) identified in the quotation, provided such Development Location(s) is located within the United States. Customer may manufacture, brand and distribute such Customer's Product(s) worldwide to its End-Users.

3.2.3. **Internal Tool License:** Customer may use the Development Tools solely: (a) for internal business purposes and (b) on the specified number of computer work stations and sites. Development Tools are licensed on a per-seat basis and shall not be distributed to others or delivered in Customer's Product(s).

3.2.4. **Evaluation License:** Mentor Graphics may from time to time, at its sole discretion, lend Software to Customer. For each loan, Mentor Graphics will identify in writing the quantity and description of Software loaned, the authorized location and the term of the loan. Mentor Graphics will grant to Customer a temporary license to use the loaned Software solely for Customer's non-production internal evaluation. Customer shall return to Mentor Graphics or delete and destroy loaned Software on or before the expiration of the loan term. Customer will sign a certification of such deletion or destruction if requested by Mentor Graphics.

3.3. Mentor Graphics' standard policies and programs, which vary depending on Products, license fees paid or services purchased, apply to the following: (a) relocation of Products; (b) use of Products, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer requests any change or enhancement to Products, or provides Product feedback, whether in the course of receiving support or consulting services, evaluating Products, performing beta testing or otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

4. **BETA CODE.**

4.1. Portions or all of certain Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.

4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

4.3. Customer agrees to maintain the Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive the termination of this Agreement.

5. **RESTRICTIONS ON USE.**

5.1. Customer may copy Software only as reasonably necessary to support the authorized use, including archival and backup purposes. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Except where embedded in Executable Code form in Customer's Product, Customer shall maintain a record of the number and location of all copies of Software, including copies merged with other software and products, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use them except as permitted by this Agreement. Customer shall give Mentor Graphics immediate written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use. Customer acknowledges that the Products provided hereunder may contain Source Code which is proprietary and its confidentiality is of the highest importance and value to Mentor Graphics. Customer acknowledges that Mentor Graphics may be seriously harmed if such Source Code is disclosed in violation of this Agreement. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any Source Code from Products that are not provided in Source Code form. Except as embedded in Executable Code in Customer's Product and distributed in the ordinary course of business, in no event shall Customer provide Products to Mentor Graphics competitors. Log files, data files, rule files and script files generated by or for the Development Tools (collectively "Files") constitute and/or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Under no circumstances shall Customer use Products or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Products, or disclose to any third party the results of, or information pertaining to, any benchmark.

5.2. Customer may not relocate, sublicense, assign or otherwise transfer this Agreement, or the licenses, rights and duties under it, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement.

5.3. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the Development Location for which support is ordered in accordance with Mentor Graphics' then current End-User Software Support Terms located at http://supportnet.mentor.com/about/legal/.

7. **THIRD PARTY AND OPEN SOURCE SOFTWARE.**

7.1. Some Products may contain code distributed under a third party license agreement. Please see the applicable Product documentation for further details. Mentor Graphics warrants that Products containing code subject to a third party license agreement do not require: (a) disclosure or distribution in Source Code form; (b) licensing for the purpose of making derivative works; or (c) redistribution at no charge, as a condition for use, modification and/or distribution of such code. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY WITH RESPECT TO THE FOREGOING LIMITED WARRANTY UNDER THIS SECTION 7 SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) TERMINATION OF THIS LICENSE AND REFUND OF THE LICENSE FEE RECEIVED FROM CUSTOMER UPON RETURN OF PRODUCT TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF PRODUCT THAT DOES NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT.

7.2. If Customer uses open source software in conjunction with Products, Customer will ensure that Customer's use does not: (a) impose, or purport to impose, obligations upon Mentor Graphics with respect to Products or (b) grant, or purport to grant, to any third party any rights to or immunities under Mentor Graphics' proprietary and other rights in Products. For purposes of this Agreement, open source software means software available for use, modification and distribution that is licensed under terms that require the licensee to make the licensee's modifications to the open source software or any software that the licensee "combines" with the open source software freely available to others in Source Code form. Customer may not combine Products with software licensed under the GNU General Public License ("GPL") in any manner that could cause, or could be interpreted or asserted to cause, the Software or any modifications to the Software to become subject to the terms of the GPL. The provisions of this Section 7 shall survive the termination of this Agreement.

8. **LIMITED WARRANTY.**

8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Software, when properly installed, and other Products will substantially conform to the functional specifications set forth in the applicable user manual and/or specification. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Products under the applicable Order and does not renew or reset, by way of example, with the delivery of (a) Software

updates or (b) authorization codes. This warranty shall not be valid if Software has been subject to misuse, unauthorized modification or improper installation. In addition, all third party software is subject to the manufacturer's original warranty as described in the manufacturer's license agreement, and is not included in this warranty. Such third party products are identified with an asterisk in the applicable quotation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF PRODUCT TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF PRODUCT THAT DOES NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS WHICH ARE LICENSED AT NO COST; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE TO CUSTOMER AND DO NOT APPLY TO ANY END-USER. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, WITH RESPECT TO PRODUCTS OR OTHER MATERIAL PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

9. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE PRODUCT OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **HAZARDOUS APPLICATIONS.**

10.1. Customer acknowledges that Mentor Graphics is licensing Products to Customer for the purpose of reducing the design and implementation time that would otherwise have been required in making Customer's designs. Customer agrees that Mentor Graphics has no control over Customer's testing or the specific applications and use that Customer will make of Products. Mentor Graphics Products are not specifically designed for use in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support systems, medical devices or other equipment in which the failure of Mentor Graphics Products could lead to death, personal injury, or severe physical or environmental damage ("Hazardous Applications").

10.2. CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING PRODUCTS USED IN HAZARDOUS APPLICATIONS AND SHALL BE SOLELY LIABLE FOR ANY DAMAGES RESULTING FROM SUCH USE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF PRODUCTS IN ANY HAZARDOUS APPLICATION. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10. THE PROVISIONS OF THIS SECTION 11 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

12. **INFRINGEMENT.**

12.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay any costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

12.2. If a claim is made under Subsection 12.1 Mentor Graphics may, at its option and expense, either: (a) replace or modify the Product so that it becomes non-infringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any fees paid, less a reasonable allowance for use.

12.3. Mentor Graphics has no liability to Customer if the claim is based upon: (a) the combination of the Product with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics or as directed by Mentor Graphics; (c) the continued use of the infringing Product when Mentor Graphics has provided Customer with a current unaltered release of a non-infringing Product in accordance with Subsection 12.2(a); (d) the use of Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or loaned

Product; (g) any third party software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers including but not limited to any SNMP Research software (SNMPv1/v2/v3); or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.

12.4. THIS SECTION 12 IS SUBJECT TO SECTION 9 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT LICENSED UNDER THIS AGREEMENT.

13. **TERMINATION AND EFFECT OF TERMINATION.**

13.1. Termination for Breach. This Agreement shall remain in effect until terminated in accordance with its terms. Upon written notice, Mentor Graphics may immediately terminate this Agreement and/or the licenses granted under this Agreement, and Customer will immediately discontinue use and distribution of Products if Customer: (a) exceeds the scope of the license granted or otherwise fails to comply with the licensing and/or confidentiality provisions of this Agreement or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license to Software granted under this Agreement at any time upon 30 days written notice to Customer if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination.

13.2. Effect of Termination. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer will discontinue use and/or distribution of Products, and either return to Mentor Graphics or destroy Products in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any Products or copies of Products in any form. Upon termination for Customer's breach, an End-User may continue its use and/or distribution of Customer's Product so long as (a) the End-User was licensed according to the terms of this Agreement and (b) such End-User is not in breach of its agreement nor a party to Customer's breach.

14. **EXPORT.** Products are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain Products, information about the Products, and direct or indirect products thereof to certain countries and certain persons. Regardless of any disclosure Customer makes to Mentor Graphics of an ultimate destination of the Products or material provided under this Agreement, or direct or indirect products thereof, Customer warrants that it and its End-Users will not export in any manner, either directly or indirectly, any such Product or material, or direct or indirect product thereof, without first obtaining all necessary approval from appropriate local and U.S. Government agencies. Customer acknowledges that the regulation of product export is in continuous modification by local governments and/or the United States Congress and administrative agencies. Customer agrees to complete all documents and to meet all requirements arising out of such modifications.

15. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.

16. **THIRD PARTY BENEFICIARY.** For any Software licensed under this Agreement and provided by Customer to End-Users, Mentor Graphics or the applicable licensor is a third party beneficiary of the agreement between Customer and End-User. Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.

17. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Products. With prior written notice and during Customer's normal business hours, an internationally recognized accounting firm chosen by Mentor Graphics, shall have the right to review Customer's records, accounts and sublicensing documents deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FLEXlm or FLEXnet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all Customer information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. Such license review shall be at Mentor Graphics' expense unless it reveals a material underpayment of fees of five percent or more, in which case Customer shall reimburse Mentor Graphics for the costs of such license review. Customer shall promptly pay any such fees. If the license review reveals that Customer has made an overpayment, Mentor Graphics has the option to either provide the Customer with a refund or credit the amount overpaid to Customer's next payment. The provisions of this Section 17 shall survive the termination of this Agreement.

18. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of the Mentor Graphics intellectual property rights licensed under this Agreement are located in Ireland and the United States. To promote consistency around the world, disputes shall be resolved as follows: This Agreement shall be governed by and construed under the laws of the State of Oregon, USA, excluding choice of law rules, if Customer is located in North or South America, and the laws of Ireland if

Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia (except for Japan) arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. This section shall not restrict Mentor Graphics' right to bring an action against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

19. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

20. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. This Agreement may only be modified in writing by authorized representatives of the parties. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 091110, Part No. 242358